Programming assignment #2

Dylan Stewart

Computer Engineering Major

ECE 407

Dstew002@odu.edu

703-209-0646

Site Location: Virginia Beach, VA

Due by: 3/2/21 at 12pm (noon) EST

Introduction

This assignment was to add simple functionality to an already made Pong game by adding sounds and editing visual characteristics of the game to make it more appealing and user friendly. In this assignment we had to first look over the, already written, code and understand what was going on and how it was generally working, then we had 9 main tasks to complete:

1. Generate a sprite font for the scores and add it to the Content Pipeline. The recommended font size is 70, but you can certainly use other font sizes. Use Bold style. Set the maximum score to be 100.
2. Replace the ball texture with a ball image created by yourself or downloaded from Internet. You may need to convert the image format from .gif to .png, since .gif is not supported by XNA.
3. Add a background image (texture) to the game. The background can simulate the table tennis table or other things.
4. Add sound effects when the ball is bounced by the paddle or when the ball hits the left or right border. Some sound effects can be found at this website: http://www.pacdv.com/sounds/interface_sounds.html. Use the SoundEffect class for this task.
5. Add background music (using .wma or .mp3 formats) to the game. Set the play mode to repeat so that the background music is continuously played. Use the MediaPlayer and Song classes for this task.
6. Handle the following keyboard inputs.
    a. Reset the game using "R".
    b. Exit the game using "Q" or "ESC".
    c. Toggle (pause and resume) the background music using "P".
7. Randomize the initial speed of the ball in the Game1. ResetGame() method so that the x component of the speed is in the intervals [4, 6] or [-6, -4] and the y component is in the intervals [3, 5] or [-5, -3]. Use the Random class for this task.
8. Display a text that displays your name. You need to generate a new sprite font.

These tasks were all handled and carried out efficiently and sometimes even exceeding the basic expectation of the tasks at hand. All Tasks that had questions are answered at the bottom of the Results section.

Program Design

For Doxygen, summaries for each task have been added. My .chm file is located inside Pong Base and is called Project_2.chm, my Doxygen configuration file is also located inside Pong Base and is called Doxygen Config. The Doxygen documentation results are inside Pong Base->Documentation.

Results

For this Assignment I got started by looking over all the code and figured out what each function's purpose was and how it was used in this program. I then started with the easy tasks that I thought would be simple to complete, like changing the maximum value from 10 to 100 and create a SpriteFont for the score. I completed these by first changing:

"`if (m_Score1 > 9 || m_Score2 > 9)`"

To:

"`if (m_Score1 > 99 || m_Score2 > 99)`"

Inside MoveBall() function. This changed the maximum value at which the program would reset, I also had to change how the score was calculated because in the originally code it was taking the modulus 10 of the score passed, i.e score % 10, and I didn't want that since it wouldn't show any values above 10, so I removed that and just printed out the score in string format.

My next task was to replace the ball with a different, round, texture that was more realistic to the user, I got this image by searching online and eventually finding a good one on imgbins.com and was able to download it and use it for this assignment. This was fairly simple because I was able to download it and just point the content loader to a different file and it worked perfectly. I did tweak the ball size from 15.0f to 17.0f just to make it more visible because it was smaller than I would have preferred to play with.

The next task involved replacing the background from the classic black background to a texture. So I searched the internet and found a great background that looks like a typical ping pong table and I was able to load that in using the same method as done with the ping pong ball, by making it a Texture2D and then using spriteBatch.Draw to draw it in the back. I did have to scale it in order to make it fit the whole screen evenly. The code to draw it in is inside the Render() function:

"`spriteBatch.Draw(Content.Load<Texture2D>(@"media/table_tennis"), Vector2.Zero, null, Color.White, 0, Vector2.Zero, new Vector2(1.65f, 2.0f), SpriteEffects.None, 0.9f);`"

Next, I had to add sound effects to the ball hitting the wall and the ball hitting the paddle. I looked through the website that was provided, http://www.pacdv.com/sounds/interface_sounds.html, and I really liked the sounds 23 and 24 because they were quick and the user could differentiate between the two easily. I chose 23 to be the sound of a ball hitting the side wall and scoring, and I chose 24 to be the sound of a ball hitting a paddle and getting deflected. It was easy to incorporate the SoundEffects class to this project, I just had to go into MonoGame and change the processor from Song to SoundEffects and shown in Figure 1(before) and Figure 2(after).

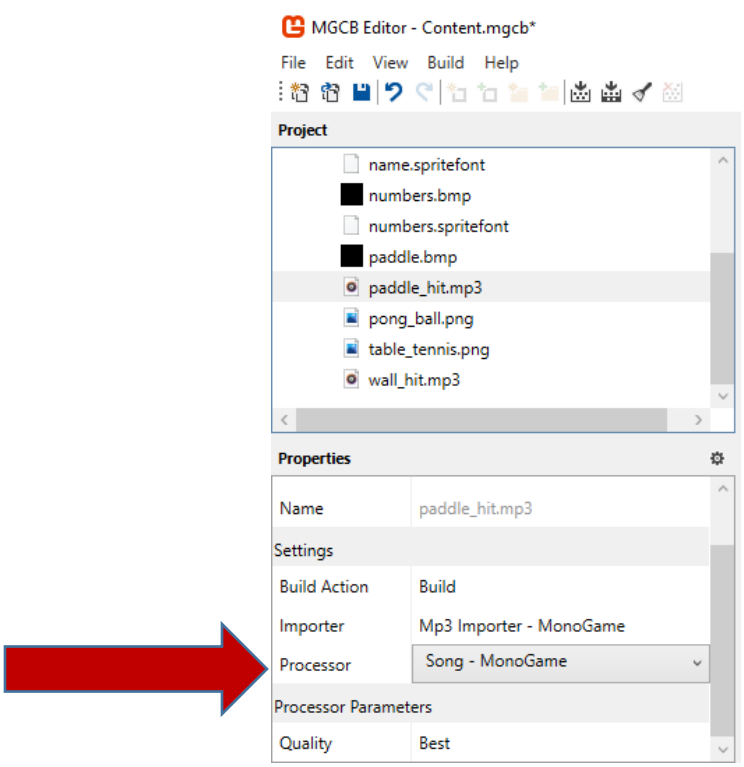

Figure 1: Before Adjustment.

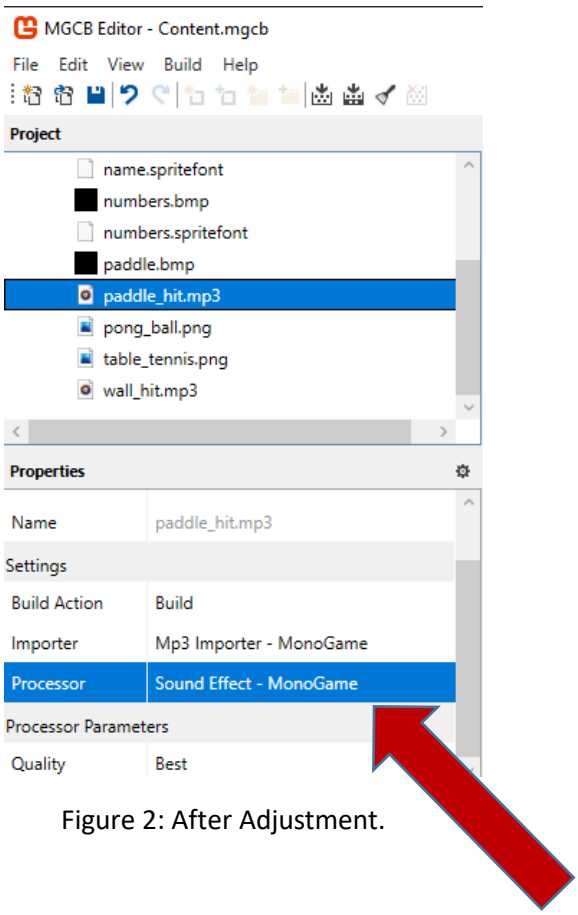

Figure 2: After Adjustment.

I imported the sounds by adding these couple lines:

Inside public class Game1:

```
// sounds
SoundEffect wallBounce;

SoundEffect paddleBounce;
```

And inside LoadContent():

```
// load sound files from disk
wallBounce = Content.Load<SoundEffect>(@"media/wall_hit");

paddleBounce = Content.Load<SoundEffect>(@"media/paddle_hit");
```

And I edited the code to play the sound effects, like so:

```
// did ball hit the paddle from the front?
if (CollisionOccurred())
{
    // reverse hoizontal direction
    m_ball.DX *= -1;
```

```
    //play sound effect
    paddleBounce.Play();

    // increase the speed a little.
    m_ball.DX *= 1.15f;
}

// actually move the ball
m_ball.X += m_ball.DX;
m_ball.Y += m_ball.DY;

// did ball touch top or bottom side?
if (m_ball.Y <= 0 ||
    m_ball.Y >= SCREEN_HEIGHT - m_ball.Height)
{
    // reverse vertical direction
    m_ball.DY *= -1;
}


// did ball touch the left side?
if (m_ball.X <= 0)
{
    // at higher speeds, the ball can leave the
    // playing field, make sure that doesn't happen
    m_ball.X = 0;

    //play sound effect
    wallBounce.Play();

    // increment player 2's score
    m_Score2++;

    // reduce speed, reverse direction
    m_ball.DX = 5.0f;
}

// did ball touch the right side?
if (m_ball.X >= SCREEN_WIDTH - m_ball.Width)
{
    // at higher speeds, the ball can leave the
    // playing field, make sure that doesn't happen
    m_ball.X = SCREEN_WIDTH - m_ball.Width;

    //play sound effect
    wallBounce.Play();

    // increment player 1's score
    m_Score1++;

    // reduce speed, reverse direction
    m_ball.DX = -5.0f;
}

// reset game if a player scores 10 goals
/*
 * EDIT: Changed max score to 100 from 10 per Task 2 in Homework Assignment.
```

```
             */
            if (m_Score1 > 99 || m_Score2 > 99)
            {
                ResetGame();
            }

            // did ball hit the paddle from the front?
            if (CollisionOccurred())
            {
                // reverse hoizontal direction
                m_ball.DX *= -1;

                //play sound effect
                paddleBounce.Play();

                // increase the speed a little.
                m_ball.DX *= 1.15f;
            }
```

The next task, I had to add some background music for the user to listen to while playing the game. This was probably one of the harder ones, but most rewarding, because I wanted to get creative and add multiple songs so that the user isn't listening to one song on repeat so I first looked into using a SongCollection to store all the songs and play them, but I found out later on that that would be a lot harder than I thought to implement it. So, I decided to using a list of Songs and declared it by stating, List<Song> songlist = new List<Song>();. This made it easy because to add a song to the end of the list all I had to do was use the .Add() operator. To start, I implemented two Song variables called background_1 and background_2 and then I linked them to the two song files that I wanted to play inside of the LoadContent() function, like so:

```
            background_1 = Content.Load<Song>(@"media/background_choice_1");
            // found at: https://www.bensound.com/ | song name: Memories.
            background_2 = Content.Load<Song>(@"media/background_choice_2");
        // found at: https://mixkit.co/free-stock-music/ | song name: Games Worldbeat.
            songlist.Add(background_1);
            songlist.Add(background_2);
            // Note: all songs are Copyright Free.


            //MediaPlayer.IsRepeating = true; //created my own version of this down in
the Update() Function.
            MediaPlayer.Volume = 75;
            MediaPlayer.Play(songlist[mediaCount]);
            b_musicPlaying = true;
```

Here you can see that I set each Song variable to the mp3 file, using Content.Load<Song>, and then I add them to the songlist and I start playing it. The mediaCount variable counts how many times any song has been played and we use this further down in the program, but here it is zero and will always be zero on start, when LoadContent() is called. Here I also commented out the IsRepeating operator because I have added my own way of looping, which is scalable with this program and allows for multiple songs to be added on afterwards. Here is my code for looping the songs:

```
      // My own version of MediaPlayer.IsRepeating, where it can play multiple songs
(one after the other) and this is scalable to allow for more songs to be added in the
future.
            // to add more songs, simply add it to the end of the List named songlist
inside LoadContent().
            if(MediaPlayer.State == MediaState.Stopped)
            {
                mediaCount++;
                MediaPlayer.Play(songlist[mediaCount % (songlist.Count)]);

            }
```

Here I check to see if the current song has stopped playing by checking the MediaPlayer.State, and if it is then we increment the mediaCount variable, so for the first iteration it would go from 0 to 1, and then we play the next one on the songlist by doing a modulus operation on the mediaCount from the size of the songlist. So, in my case we have 2 songs in the list and so no matter how high the mediaCount counts to, it will always alternate between a 0 and 1 which works with the songlist List. It is more complex than just using the .IsRepeating operator, but I feel like it is worth it to add more flexibility and scalability in the long run, if this were an actual program for a company.

For the next task, I had to handle the key presses, “R” for Reset, “Q” or “Esc” for quit/exit and “P” for pausing the music. I first initiated this by using a simple if statement that stated:

```
if (newState.IsKeyDown(Keys.P)){..}
```

But then I realized that it would say I’m clicking it a lot even though I only clicked it once. This was because the program was moving so fast that, even though to me I was only clicking it once, the computer would read it 10-15 times in that quick span of me just clicking it, so I added another KeyState variable called oldState and this would hold the previous value of the newState before the function is done completing. Then, I was able to implement this efficiently using the new if statement:

```
if (newState.IsKeyDown(Keys.P) && oldState.IsKeyUp(Keys.P))
```

And this worked and only read one key click rather than the multiple like before and so, I implemented this in all the KeyState checks inside the Update() function, like so:

```
newState = Keyboard.GetState();
// Allows the game to exit
```

```
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
                || (newState.IsKeyDown(Keys.Q) && oldState.IsKeyUp(Keys.Q))
                || (newState.IsKeyDown(Keys.Escape) && oldState.IsKeyUp(Keys.Escape)))
                this.Exit();

            // Reset game if "R" is pressed
            if (newState.IsKeyDown(Keys.R) && oldState.IsKeyUp(Keys.R))
                ResetGame();

            // Pause music if "P" is pressed
            if (newState.IsKeyDown(Keys.P) && oldState.IsKeyUp(Keys.P))
            {
                if (!b_musicPlaying)
                {
                    MediaPlayer.Resume();
                    b_musicPlaying = true;
                }
                else if (b_musicPlaying)
                {
                    MediaPlayer.Pause();
                    b_musicPlaying = false;
                }
            }

            // My own version of MediaPlayer.IsRepeating, where it can play multiple
songs (one after the other) and this is scalable to allow for more songs to be added in
the future.
            // to add more songs, simply add it to the end of the List named songlist
inside LoadContent().
            if(MediaPlayer.State == MediaState.Stopped)
            {
                mediaCount++;
                MediaPlayer.Play(songlist[mediaCount % (songlist.Count)]);
            }

            // update the ball's location on the screen
            MoveBall();

            // update the paddles' locations on the screen
            MovePaddles();

            base.Update(gameTime);
            oldState = newState;
```

Next task required a bit of randomization. I had to randomize the x and y direction of the ball everytime it was reset, with strict parameter of x: [4 through 6] and [(-6) through (-4)] | y: [3 through 5] and [(-5) through (-3)]. I initially knew I had to start in the ResetGame() function and add a couple random variables. I also had to add 2 other variables to check whether I should make them negative. So, to implement this I came up with this solution:

```
            Random r_x = new Random();
            Random r_y = new Random();
            Random neg_x = new Random();
            Random neg_y = new Random();
```

```
            int b_neg_x = neg_x.Next(2); //between 0 and 1, if 0: dont use negative, if
1: use negative.
            int b_neg_y = neg_y.Next(2); //between 0 and 1, if 0: dont use negative, if
1: use negative.

            // reset scores
            m_Score1 = 0;
            m_Score2 = 0;

            // place the ball at the center of the screen
            m_ball.X =
                SCREEN_WIDTH / 2 - m_ball.Width / 2;
            m_ball.Y =
                SCREEN_HEIGHT / 2 - m_ball.Height / 2;

            // set a speed and direction for the ball
            //m_ball.DX = 5.0f;
            //m_ball.DY = 4.0f;
            m_ball.DX = (float)r_x.NextDouble() * 3 + 4;
            m_ball.DY = (float)r_y.NextDouble() * 3 + 3;
            if (b_neg_x == 1) m_ball.DX *= -1;
            if (b_neg_y == 1) m_ball.DY *= -1;
```

Here you can see the Random variable are being used to get the .NextDouble(), which returns a value from 0 to 1 and in order to shift it to the place I needed it, I needed to multiply by the size of the strict parameter, for x: 4 through 6 is 3 and for y: 3 through 5 is 3 as well, and then I needed to add how much I wanted to shift over, for x is was to 4 and for y it was to 3. And this should give us an accurate float value between the set parameters. And to differentiate between whether a number should be negative or not, I randomized an integer between 0 and 1, if 0 then I would not use the negative value, but if 1 then I would multiply it by -1.

For the final task, this was quite easy. I simply added a SpriteFont description to the Monogame content and left it as default, then imported into the game like the score, inside LoadGameGraphics() function, and then used spriteBatch.DrawString to draw the text onto the screen inside Render(), like so:

```
//draw my name
spriteBatch.DrawString(m_name, "Created by Dylan Stewart", new
Vector2(SCREEN_WIDTH - 190, SCREEN_HEIGHT - 25), Color.White);
```

Here I subtracted 190 from the Screens width and 25 from the Screens height to account for the size of the text and place it right at the bottom right of the screen.

For the questions asked in Tasks 10 -12:

10 - This is a problem because we are already moving the ball by doing “m_ball.X += m_ball.DX” before this so this could allow for the ball to look like it’s getting through the paddle a little bit and there is nothing inside the CollisionOccured() function that is stopping it from going past the paddles boundaries besides just switching it by making it negative.

11 - To make this more realistic to the user, we could simply check for collision before and after we do any moving to the ball. I have implemented this into my program. This allows for the function to initially check if the ball has made contact and if so, then divert it and then after we move the ball is checks again to see if it has made contact. This is a two-factor check to make sure we don't pass the paddle.

12 - A better implementation would be to add a pause screen for the users so that they can pause and take a break and not worry about the game continuing while they're away. Also, another implementation for the future would be to increase the paddle's speed as the ball's speed increases, because I found that too many times once the ball got to a certain speed, my paddle couldn't even keep up with it and would be too slow to get to it. We could increase the speed of the paddle but set a max speed so that the user isn't just flying from one end of the screen to the other with one short press.

Conclusion

To conclude this project, I accomplished so much in this project. I learned a lot about how to add sound effects and songs and manipulate them and experiment with them to make it more user-friendly and I really enjoyed seeing my progress come to life. Most of my difficulties simply came from not knowing how to implement certain things, like the SongCollection that I previously talked about. It can only be fixed by gaining more experience and practicing more. Overall, I really enjoyed this project and it makes me excited for what to come in this class!